Új eredemények a szoftvertermék-minőség mérésben, és azok alkalmazásai a szoftverevolúció során

Hegedűs Péter

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2014

Témavezető:

Dr. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem

Informatikai Doktori Iskola

Bevezetés

A napjainkra jellemző szoftverrendszerektől való egyre nagyobb függés (gondoljunk csak a repülés irányító szoftverekre vagy nukleáris létesítmények vezérlő rendszereire) megkerülhetetlen kutatási területté tette a szoftverek minőségének és megbízhatóságának elemzését. Sajnos a szoftverek minősége olyan összetett fogalom, amelynek teljes feltérképezése és modellezése nagyon nehéz feladat.

Jelen munka a szoftverek minőségének egyik aspektusára, a karbantarthatóságra összpontosít. Az ISO/IEC 9126 szoftverminőség szabvány [15] (utódja az ISO/IEC 25010 [16]) definíciója szerint a karbantarthatóság a "szoftverrendszer azon képessége, hogy milyen könnyű azt módosítani". A definíció alapján máris világossá válik, hogy a karbantarthatóság közvetlen kapcsolatban áll a rendszer működésének megváltoztatásához szükséges költségekkel, és hogy szorosan kapcsolódik a rendszer forráskódjához. Mint ilyen, a karbantarthatóság a "szoftver egészségének" (integritásának) egy jó mutatója lehet, ráadásul szoros összefüggésben áll a hibák rendszer forráskódjába történő bekerülésének valószínűségével, azaz tekinthetünk rá úgy is, mint a szoftver *műszaki minőségére*. Ezáltal a karbantarthatóság a modern szoftveripar egyik központi elemévé vált, és számos javaslat és ellenjavallat született azzal kapcsolatban, hogy hogyan írjunk jól karbantartható rendszereket (például a tervezési minták [18] vagy refaktoring [7] alkalmazása, vagy a tervezési ellenminták [1] elkerülése).

Mindazonáltal jelenleg a szoftveriparban a karbantarthatóságot sokszor háttérbe szorítja az új funkciók fejlesztése, amelyek üzleti értéke sokkal nyilvánvalóbb, legalábbis rövid távon. Mivel a karbantarthatóság fenntartása a megfelelő irányelvek követésével, illetve a nem javasolt konstrukciók elkerülésével szintén erőforrást igényelnek, ám nem hoznak rövid távon külön bevételt, így az üzleti szereplők hajlamosak azt figyelmen kívül hagyni. A karbantarthatóság mibenlétének mélyebb megértése által, illetve a különböző kódolási gyakorlatokhoz és a hosszú távú fejlesztési költségekhez való viszonyának feltárásával rávilágíthatunk a karbantarthatóság fenntartásának megtérülésére, ami az üzleti felek számára is sokkal vonzóbbá teheti azt. Az üzleti szereplők meggyőzése mellett azonban az is nagyon fontos, hogy (i) a fejlesztők, akik végül elvégzik a konkrét programozási feladatokat megkapjanak minden szükséges alacsony szintű információt ahhoz, hogy a rendszerek karbantarthatóságát ténylegesen javítani tudják; illetve (ii) megbizonyosodjanak arról, hogy a karbantarthatóság javítására tett erőfeszítéseiknek valóban van hasznuk (például a rendszerben kevesebb kiadás utáni hiba keletkezik, vagy a jövőbeni fejlesztéseket sokkal kisebb ráfordítással tudják elvégezni).

Disszertációm a fent vázolt problémák lehetséges megoldásaira koncentrál. Az értekezés három fő eredménye a következő:

I. **Bevezetünk egy olyan magas szintű mérőszámot a rendszerek karbantarthatóságának jellemzésére, amely újszerű megközelítést használ a jelenlegi módszerekhez képest, és értékes információval szolgál a programozói tudással nem rendelkező személyek számára is (mint például a menedzserek). A módszert egy Java nyelvre készített prototípus modellen validáltuk, illetve létrehoztunk egy C# modellt is, amelyet valós, ipari környezetben használtunk fel.**

II. **Bemutatunk egy olyan módszert, amivel hasznos információt kaphatunk a karbantarthatóságról a forráskód elemek szintjén, ami aztán felhasználható a rendszer karbantarthatóságának javításához, vagy segítséget nyújthat a szoftver evolúció során felmerülő feladatok elvégzésében, mint például a tesztelési erőforrások összpontosítása, a kód átvizsgálás célpontjainak kiválasztása vagy a költségek becslése.**

III. Számos esettanulmányt végeztünk annak érdekében, hogy bemutassuk az újonnan kidolgozott karbantarthatóság modellező módszerekben rejlő lehetőségeket a hibák helyének beazonosítására, a fejlesztési költségek becslésére és a kódolási gyakorlatok (mint például a tervezési minták alkalmazása) karbantarthatóságra gyakorolt hatásának feltárására. Utóbbihoz egy általános, visszatervező eszközök kiértékelését lehetővé tévő benchmark használatát is javasoltuk.

I. Rendszer szintű karbantarthatóság modellezése

Ezen tézispont eredményei a szoftver termékek minőségének rendszer szintű modellezéséhez kapcsolódnak.

Egy valószínűség számításon alapú karbantarthatósági modell és validációja

A jelenleg létező karbantarthatósági modellek egy felmérésben [5] kimutatott gyakori hátrányainak kiküszöbölése céljából bevezetünk egy valószínűség számításon alapuló megközelítést [4] az ISO/IEC 9126 [15] és az ISO/IEC 25010 [16] szabványokban definiált magas szintű minőségi jellemzők kiszámítására, amely figyelembe veszi a szakértők tudását, és kezeli a minőség definíciójából adódó bizonytalanságot is. A módszer úgynevezett "jóság" függvényeket használ, amik a küszöbérték alapú módszerek folytonos általánosításai. A magas szintű minőségi jellemzők kiszámítása egy irányított körmentes gráf alapján történik, amelynek csomópontjai megfelelnek az egyes belső (alacsony szintű), illetve külső (magas szintű) minőségi mutatóknak. A valószínűség számításon alapuló statisztikai összegző algoritmus egy ún. benchmarkot használ a minősítéshez, ami egy 100 nyílt forrású és ipari rendszer forráskód metrikáit tartalmazó referencia adatbázis. Két, az új valószínűség számításon alapuló minőségi modellel megvizsgált Java rendszer alapján megállapítottuk, hogy a modell eredményeiben történő változások jól tükrözték a különböző fejlesztési tevékenységeket, például a fejlesztési fázisok alatt a rendszer minősége általában csökkent, míg karbantartás közben (pl. szerkezeti javítás) a minőség általában nőtt. E mellett a modell által számított "jóság" értékek magas korrelációt mutattak a szakértői kiértékelés eredményeivel is.

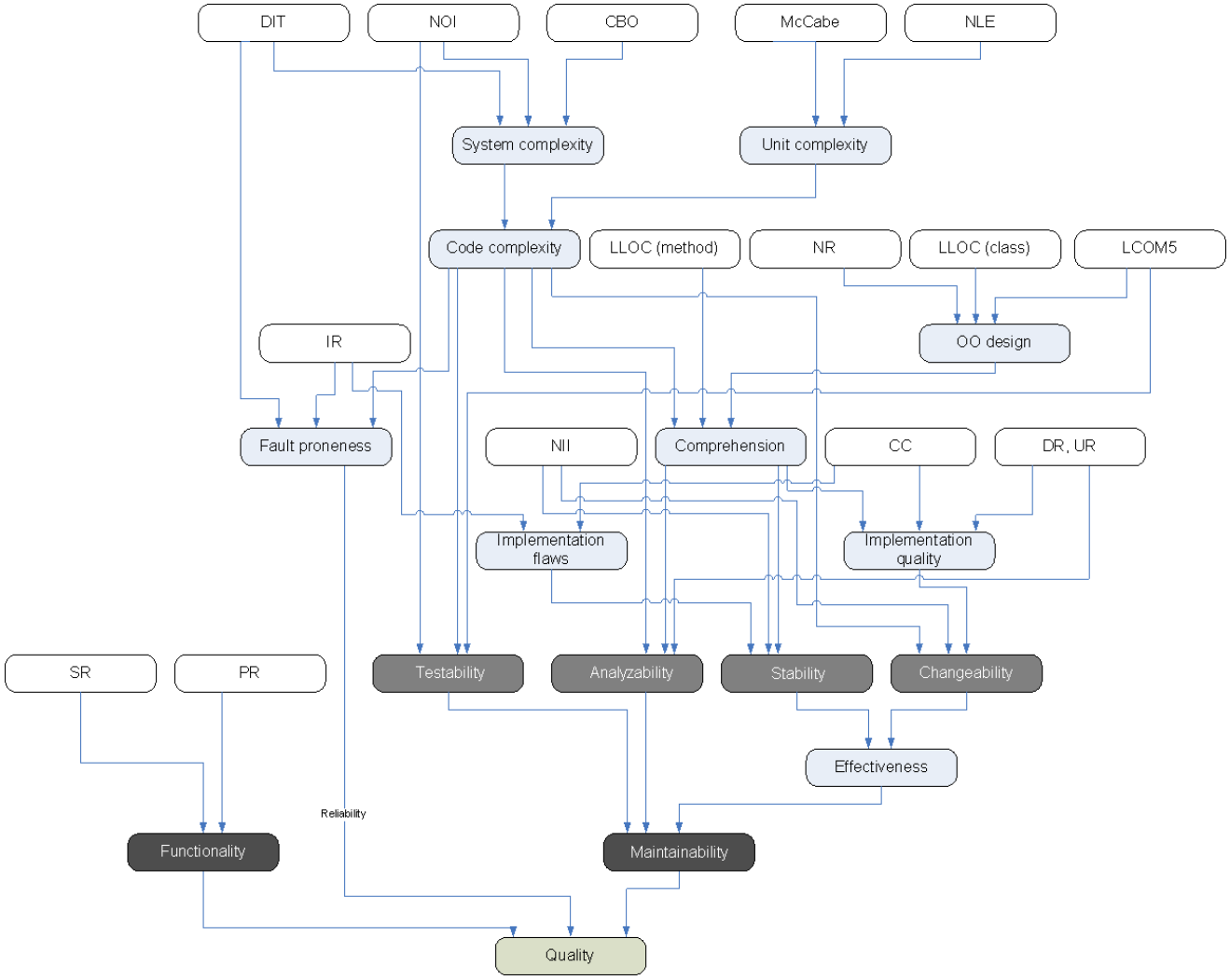
Verzió	Változtathatóság	Stabilitás	Elemezhetőség	Tesztelhetőség	Karbantarthatóság
REM v0.1	0.625 (0.7494)	0.4 (0.7249)	0.675 (0.7323)	0.825 (0.7409)	0.625 (0.7520)
REM v1.0	0.6 (0.7542)	0.65 (0.7427)	0.75 (0.7517)	0.8 (0.7063)	0.75 (0.7539)
REM v1.1	0.6 (0.7533)	0.66 (0.7445)	0.7 (0.7419)	0.66 (0.6954)	0.633 (0.7402)
REM v1.2	0.65 (0.7677)	0.65 (0.7543)	0.8 (0.7480)	0.775 (0.7059)	0.7 (0.7482)
Korreláció	**0.71**	**0.9**	**0.81**	**0.74**	**0.53**
System-1 v1.3	0.48 (0.4458)	0.33 (0.4535)	0.35 (0.4382)	0.43 (0.4627)	0.55 (0.4526)
System-1 v1.4	0.6 (0.4556)	0.55 (0.4602)	0.52 (0.4482)	0.4 (0.4235)	0.533 (0.4484)
System-1 v1.5	0.64 (0.4792)	0.64 (0.4966)	0.56 (0.4578)	0.46 (0.4511)	0.716 (0.4542)
Korreláció	**0.87**	**0.81**	**0.94**	**0.61**	**0.77**

1. táblázat. A fejlesztők véleményének átlagos értéke a karbantarthatóság és jellemzői esetében

Az 1. táblázat a fejlesztői minősítés értékeinek átlagát mutatja be a két kiértékelt rendszer különböző verzióira a modellünk által adott értékeléssel együtt a zárójelekben. Ahogyan az látható is, a két érték magas korrelációt mutat, ami azt jelenti, hogy azok többnyire egyformán változnak.

Egy karbantarthatósági modell C# programokhoz

A Java modell mellett kidolgoztunk egy karbantarthatósági modellt C#-ra [10] is az egyik ipari partnerünkkel közösen, amelynek munkatársai nagyon elégedettek voltak az előálló eredményekkel. A modellt arra használtuk fel, hogy megbecsüljük a cég több, mint 300 szoftver komponensének minőségét, és felállítsunk egy rangsort közöttük. A megalkotott modellt az 1. ábra szemlélteti.



1. ábra. A C# karbantarthatósági modell

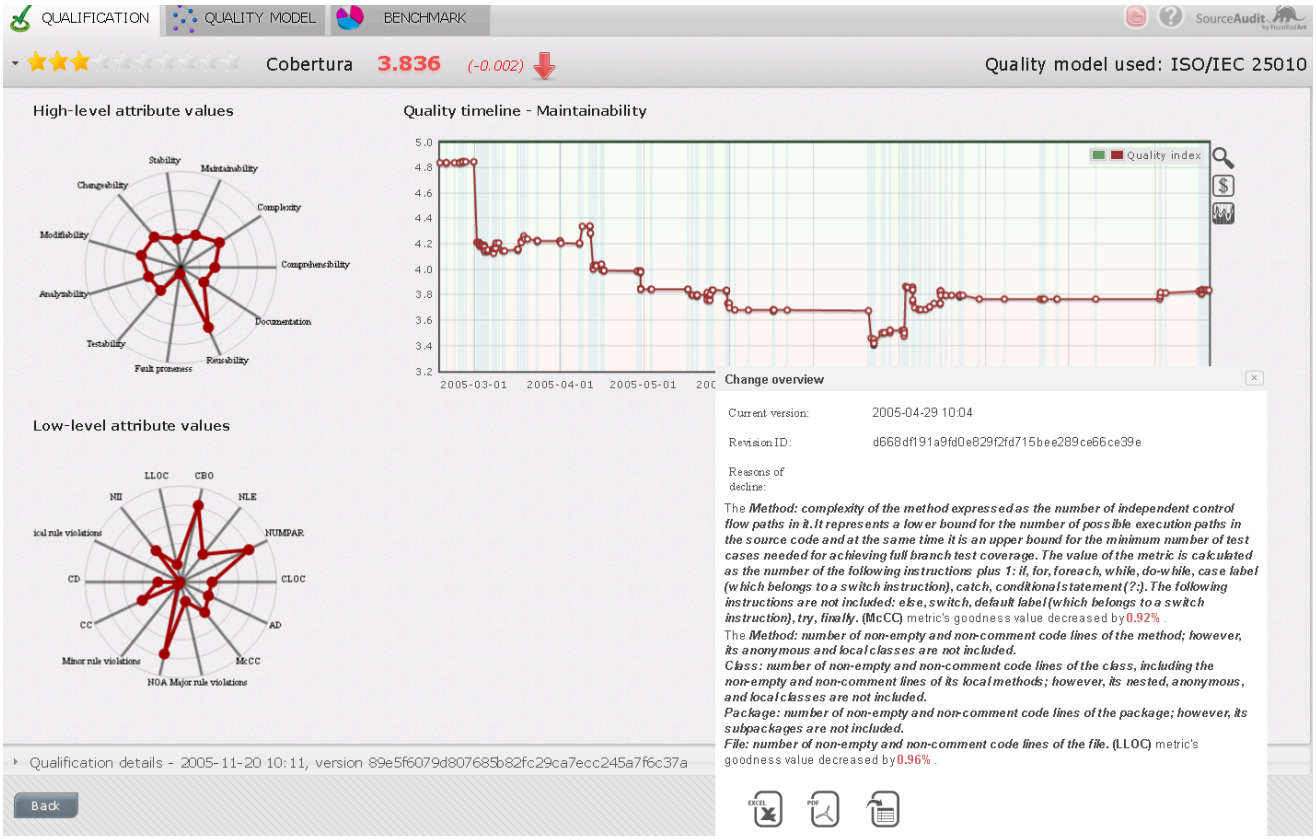
A modellünk által előállított eredményeket összevetettük a fejlesztők személyes véleményével is 10 kiválasztott komponensen, és bár az átlagos emberi szavazatok magasabb értéket mutattak a modell által becsült értékeknél (ahogyan az a 2. táblázatból is látszik), a Pearson korreláció analízis 0.92-es értéket mutatott 0.01-es szignifikancia szint mellett, amely erős összefüggést jelez a két adatsor között.

Karbantarthatóság	0.311	0.261	0.261	0.261	0.26
Szakértői vélemény átlag	0.56	0.48	0.473	0.53	0.47
Karbantarthatóság	0.26	0.221	0.221	0.216	0.178
Szakértői vélemény átlag	0.49	0.4	0.44	0.45	0.3

2. táblázat. A karbantarthatósági értékek és a szakértői vélemények átlagai 10 vizsgált komponensre

A módszer megvalósítása és kiértékelése

Az új valószínűség számítási módszeren alapuló megközelítés megvalósításaként létrehoztuk a *SourceAudit* [3] nevű eszközt, amely a *QualityGate* folyamatos minőség monitorozó keretrendszer részét képezi (az eszköz egyik fő felülete a 2. ábrán látható). Az eszközt sikerrel hasznosítottuk néhány magyar és nemzetközi kutatás és fejlesztés projekt keretein belül, valamint az a FrontEndART Kft. hivatalos kereskedelmi termékévé vált. Továbbá a megvalósított eszközt kiértékeltük és összehasonlítottuk számos hasonló eszközzel a szoftver minőség mérés területén belül [5].



2. ábra. Egy rendszer minőségi elemzésének részletei

A szerző hozzájárulása az eredményekhez

A szerző végezte a meglévő gyakorlati minősítő modellek felmérését, kiértékelését és elméleti hátterük feltérképezését. Ő hajtotta végre az új valószínűség számításon alapuló minősítő módszer empirikus validációját, az eredmények kiértékelését, és implementálta a validáció elvégzését segítő prototípus eszközöket. A teljes C# minősítő modell a szerző munkája, egészen pontosan a C# specifikus modell

megalkotása, a minősítéshez elengedhetetlen szakértői súlyok és benchmark rendszerek összegyűjtése, a szükséges eszközök implementálása, valamint a modell empirikus validációjának végrehajtása és kiértékelése. A szerző részt vett a módszert megvalósító SourceAudit eszköz megtervezésében és kifejlesztésében. Ő végezte el és értékelte ki a különböző szoftver minősítő eszközöket összehasonlító esettanulmányt.

II. Forráskód elemek karbantarthatóságának modellezése

Ezen tézispont eredményei a szoftver termékek minőségének forráskód elemek szintjén történő modellezéséhez kapcsolódnak.

Előkészítő esettanulmányok

Három nagyszabású esettanulmányt [12, 14] is elvégeztünk annak megvizsgálására, hogy mennyire alkalmasak a forráskódból kinyerhető termék metrikák a karbantarthatóság előrejelzésére a forráskód elemek szintjén. Ehhez sok szubjektív véleményt gyűjtöttünk be különböző forráskód elemek minőségi jellemzőit illetően informatikai szakemberektől, és különböző szakmai tapasztalattal rendelkező hallgatótól. Az ISO/IEC 9126-os szabványban definiált minőségi jellemzőket használtuk, és a kiértékelőknek ezeket a jellemzőket kellett értékelniük egy 0-tól 10-ig terjedős skálán (a 0 volt a legrosszabb, míg a 10-es a legjobb érték) több különböző forráskód elemre. A kiértékelők szavazatainak átlagát felhasználva, különböző gépi tanulási módszerek segítségével olyan predikciós modelleket tudtunk építeni, amelyek a forráskódból kiszámítható szoftver termék metrikák alapján közelítették az emberi véleményeket a rendszer különböző minőségi jellemzőivel kapcsolatban. Arra a következtetésre jutottunk, hogy a metrikákban megvan a lehetőség a magas szintű minőségi jellemzők emberi megítélésének közelítésére (a kiértékelők véleménye 0,5 és 2 közötti szórást mutatott a 10-es skálán). Ahogyan az a 3. táblázatban is látható, a *Változtathatóság* jellemző esetén a döntési fa alapú osztályozó algoritmus közel 77%-os pontosságot ért el (ebben az esetben a kiértékeléseket három kategóriába soroltuk).

	ZeroR	**J48 Döntési fa**	**Log. regresszió**	**Neurális háló**
Elemezhetőség	67.93%	73.68%	70.97%	70.25%
Változtathatóság	66.79%	76.65%	73.00%	74.26%
Stabilitás	70.20%	73.12%	70.55%	70.92%
Tesztelhetőség	66.55%	64.72%	69.45%	70.54%

3. táblázat. A helyesen osztályozott forráskód elemek aránya

A különböző regressziós technikák áttekintése után azt mondhatjuk, hogy azok még inkább alkalmasak a minőségi jellemzők előrejelzésére azáltal, hogy folytonos skálát használnak az osztályozó algoritmusok diszkrét kategóriáival szemben. A legjobb regresszió alapú modell a fent leírt kiértékelési adatokon 0.72-es korrelációval és 0.83-as átlagos hibával közelítette a *Karbantarthatóság* jellemzőt. A 4. táblázatban a különböző regressziós algoritmusok hatékonysága látható.

Drill-down módszer a forráskód elemek karbantarthatóságának mérésére

Az esettanulmányok során megszerzett tapasztalatok alapján kidolgoztunk egy úgynevezett "drill-down" módszert [11], amellyel felderíthetők az egyes minőségi osztályzatok kiváltó okai, és az egyes

	ZeroR		Neurális háló		Lineáris Reg.		Döntési fa	
	Átl. hiba	**Korr.**	**Átl. hiba**	**Korr.**	**Átl. hiba**	**Korr.**	**Átl. hiba**	**Korr.**
Elemezhetőség	1.201	-0.162	1.076	0.408	1.076	0.466	0.884	0.660
Változtathatóság	1.026	-0.116	1.088	0.362	0.965	0.437	0.861	0.571
Érthetőség	1.574	-0.153	1.387	0.275	1.188	0.491	1.048	0.621
Stabilitás	0.822	-0.239	0.824	0.297	0.833	0.360	0.670	0.572
Tesztelhetőség	1.189	-0.118	1.168	0.427	1.145	0.363	0.926	0.639
Karbantarthatóság	1.187	-0.122	1.193	0.587	0.909	0.615	0.831	0.723
Átlag	**1.166**	**-0.152**	**1.123**	**0.393**	**1.019**	**0.455**	**0.870**	**0.631**

4. táblázat. A vizsgált regressziós algoritmusok által elért átlagos hiba és korreláció értéke

forráskód elemekhez (mint például az osztályok és metódusok) egy relatív karbantarthatósági indexet (RMI) rendelhetünk, amely a létező megközelítésekkel ellentétben a metrikák együttes értékeit is figyelembe veszi. Ezáltal lehetőségünk nyílik az egyes forráskód elemek sorba rendezésére oly módon, hogy a karbantarthatóság szempontjából legkritikusabb elemek a lista elejére kerülnek, ami a rendszerek karbantartói számára fontos lehet az erőforrások megfelelő helyre történő kiosztásához, és a rendszer legnagyobb mértékű javításának eléréséhez minimális erőbefektetés mellett. A módszert 191 Java metódus modell által számított karbantarthatósági indexének, és a metódusok kézzel történő kiértékelésének összevetésével validáltuk a jEdit nyílt forrású szövegszerkesztő rendszeren. A metódusok kézi kiértékelése, amelyet közel 200 hallgató végezett el, 0,68-as Spearman korrelációt mutatott ($p < 0.001$) a modell alapú kiértékeléssel. A korreláció elemzés részletes eredményeit az 5. táblázat szemlélteti. A drill-down algoritmus megvalósítása bekerült a fent említett SourceAudit [3] kereskedelmi minőség monitorozó eszközbe is.

Minőségi jellemzők	**Korreláció a hallgatók véleményével (R érték)**	**p-érték**
Elemezhetőség	0.64	< 0.001
Érthetőség	0.62	< 0.001
Változtathatóság	0.49	< 0.001
Stabilitás	0.49	< 0.001
Tesztelhetőség	0.61	< 0.001
Karbantarthatóság	**0.68**	$<$**0.001**

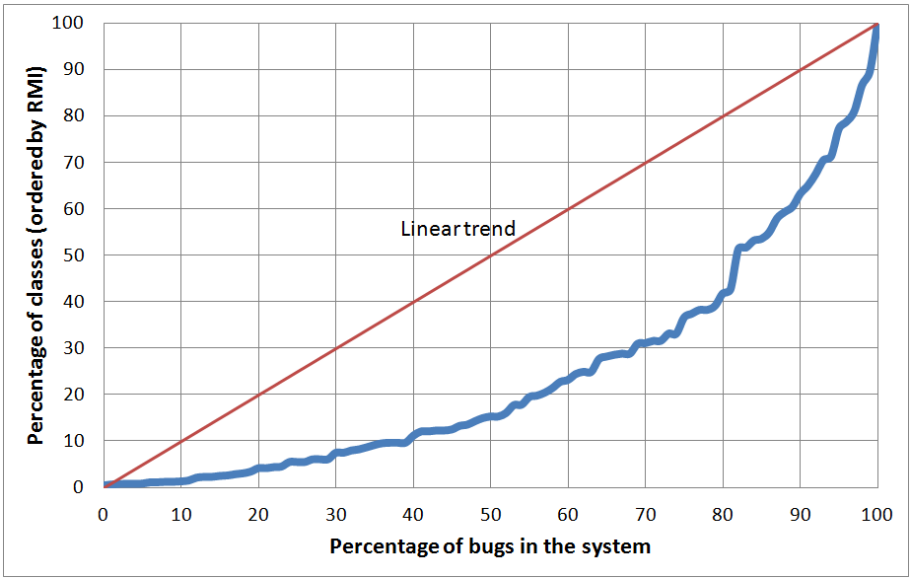
5. táblázat. Az RMI és a kézi kiértékelések közti Spearman korreláció mértéke

A szerző hozzájárulása az eredményekhez

A szerző dolgozta ki az előkészítő tanulmányok alapelveit, ő állította össze a kérdéssorokat, és alkotta meg egy web-alapú metrika kiértékelő keretrendszer alapötletét a kiértékelések begyűjtésének támogatására. Ő értékelte ki, és hasonlította össze a felmérés adatait a gépi tanuló algoritmusok eredményeivel, és ez alapján levonta a megfelelő következtetéseket. Az ő nevéhez fűződik a drill-down módszer elméleti alapjainak, valamint a módszer validációs technikájának a kidolgozása, illetve a validáció végrehajtása nyílt forráskódú rendszereken, majd az eredmények értelmezése és kiértékelése.

III. A kidolgozott módszerek alkalmazásai

Ezen tézispont eredményei a korábban bemutatott modellek, módszerek és eszközök szoftver evolúció során történő alkalmazási lehetőségeit demonstrálják.



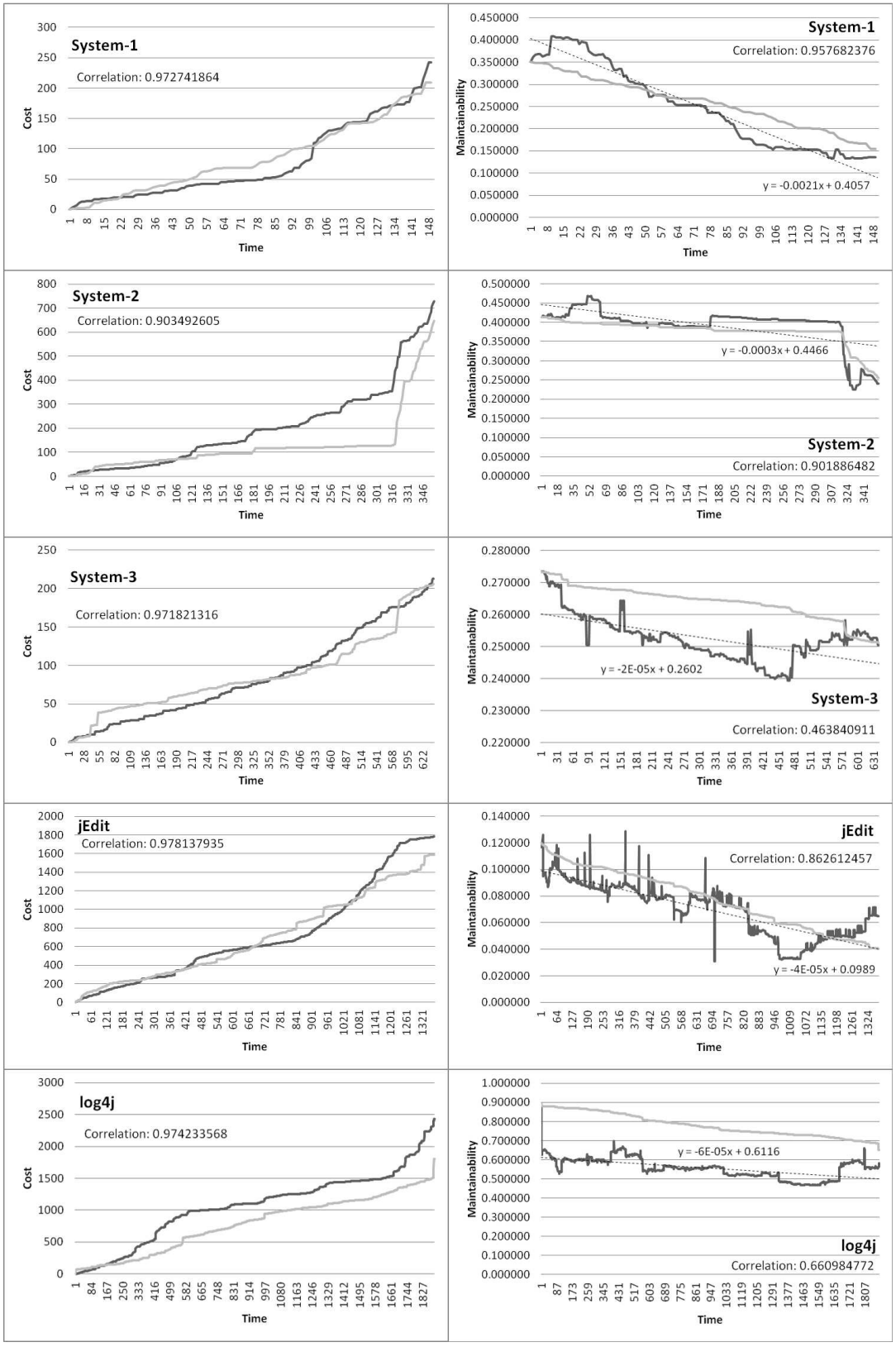
3. ábra. Különböző hiba lefedettség az osztályok RMI alapú sorrendjének arányában

A hibák helyének beazonosítása a drill-down módszer segítségével

Statisztikai módszerek segítségével megmutattuk, hogy a relatív karbantarthatósági mutató hatékony a hibára hajlamos osztályok (osztályok, amelyek sok hibát tartalmaznak) elkülönítésére azon osztályoktól, amelyek kis valószínűséggel tartalmaznak csak hibákat [17]. Egy, különböző nyílt forráskódú rendszerek 30 kiadását vizsgáló esettanulmányunk eredményei azt mutatták, hogy átlagosan a rendszerek osztályainak legrosszabb karbantarthatósági mutatóval rendelkező 30%-a tartalmazta a hibák több, mint 70%-át. A különböző hiba lefedettség értékek az RMI alapján sorba rendezett osztályok arányának függvényében a 3. ábrán látható. Az eredmények azt igazolják, hogy az osztályok karbantarthatósági index alapú sorba rendezése nagyon jó stratégia lehet a tesztelési erőforrások összpontosítására, vagy a kód átvizsgálás célpontjainak kiválasztására.

Egy karbantarthatóság alapú költségbecslő modell és annak validációja

Egy költségbecslő modell is kidolgozásra került [2], amely képes a rendszer karbantarthatóságának változása alapján a jövőbeni fejlesztések erőforrásigényét megbecsülni. Néhány egyszerű feltevésre alapozva, valamint a termodinamikában használatos entrópia fogalmát felhasználva megmutattuk, hogy amennyiben nem teszünk kifejezetten a kód javítására irányuló lépéseket, egy rendszer karbantarthatósága exponenciális mértékben csökken a fejlesztésekre fordított erőforrások függvényében. A karbantarthatóság és költségek közt feltárt kapcsolatot felhasználtuk két nyílt forrású, és három kereskedelmi rendszer jövőbeni fejlesztési költségeinek becsléséhez. Ennek részletes eredményei a 4. ábrán láthatóak. Az ábra bal oldali diagramjai a valós (sötét vonalak) és a modell által becsült (világos vonalak) költségeket mutatják. A jobb oldali diagramokon a sötét vonalak a karbantarthatóság értékének



4. ábra. A becsült valamint valós költségek és karbantarthatóság az idő függvényében

változását ábrázolják, míg a világos vonalak a modell által becsült karbantarthatóságot jelölik. A diagramokon feltüntettük a valós és becsült görbék közötti Pearson korrelációs értéket is. Ennek magas értéke azt támasztja alá, hogy a modell egyszerre írja le jól mind a költségeket, mind pedig a karbantarthatóságot.

A tervezési minták karbantarthatóságra gyakorolt hatásának feltárása

Az általunk javasolt modell alkalmazható a karbantarthatóság, és a különböző kódolási gyakorlatok (mint például a tervezési minták, ellenminták, kód másolatok vagy refaktorálás) alkalmazása közötti kapcsolat vizsgálatára is, hiszen ezek a kódolási gyakorlatok az általános nézet szerint vagy pozitívan, vagy pedig negatívan befolyásolják a rendszerek karbantarthatóságát. Például széles körben elterjedt nézet az, miszerint a tervezési minták alkalmazása jobb minőségű szoftvert eredményez, azonban viszonylag kevés objektív kísérleti eredmény támasztja alá, hogy használatuk tényleg előnyös lenne. Ami azt illeti, néhány tanulmány arra a következtetésre jutott, hogy a tervezési minták alkalmazása akár veszélyes is lehet [18]. A tervezési minták karbantarthatóságra gyakorolt hatásának empirikus vizsgálataként először a JHotDraw Java grafikus keretrendszer nagyjából 300 verzióját elemeztük [13], amely program szerkezete erőteljesen épít a jól ismert tervezési mintákra. Azt találtuk, hogy minden újonnan bevezetett tervezési minta növekedést okozott a JHotDraw rendszer minőségi jellemzőinek értékében, ahogy azt a 6. táblázat is szemlélteti. Ráadásul a tervezési mintában szereplő kódsorok átlagos aránya a rendszeren belül magas, 0,89-es Pearson korrelációt mutatott a becsült karbantarthatósági értékkel, 0,05-ös szignifikancia szint mellett.

Verzió (r)	**Minta**	**Minta sűrűség**	**Karbantarthatóság**	**Tesztelhetőség**	**Elemezhetőség**	**Stabilitás**	**Változtathatóság**
531	+3	↗	↗	↗	↗	↗	↗
574	+1	↗	↗	↗	↗	↗	↗
609	–1	↘	—	—	—	—	—
716	+1	↘	↗	↗	↗	↗	↗
758	+1	↗	↗	↗	↗	↗	↗

6. táblázat. A szoftverminőség jellemzőinek alakulása a tervezési minták számának változásával

A kezdeti eredmények megerősítéseként megismételtük a tanulmányt 9 különböző nyílt forráskódú rendszerrel is, ahol a rendszerekből 5 különböző mintakereső eszköz által kinyert, a DPB [6] online benchmarkban megtalálható tervezési mintákat vizsgáltunk. A minták sűrűsége az előző kísérlethez hasonlóan magas Pearson korrelációs (0,59 és 0,78 közötti) és Spearman korrelációs (0,68 és 0,82 közötti) értékeket mutatott a rendszerek karbantarthatóságával, 0,05-ös szignifikancia szint mellett. A hamis minta példányok benchmarkban található kiértékelések alapján történő kiszűrése után további, nagyjából 10%-os javulást tapasztaltunk a korrelációs értékekben.

Egy visszatervező eszközök kiértékelésére szolgáló benchmark

A hosszabb távú kutatási célunk elérése érdekében, ami nem más, mint a különböző kódolási gyakorlatok és minták karbantarthatóságra gyakorolt hatásának vizsgálata, javasoljuk a BEFRIEND (BEnchmark For Reverse engInEering tools workiNg on source coDe) [9] nevű benchmarkunk felhasználását,

amely lehetővé teszi a különböző visszatervező eszközök eredményeinek összehasonlítását és kiértékelését. A rendszer a DEEBEE [8] tervezési minta benchmark általánosításának tekinthető. A BEFRIEND segítségével nagy mennyiségű pontos elemzendő adatot gyűjthetünk a különböző kódolási gyakorlatok és minták karbantarthatóságra gyakorolt hatásának felderítéséhez.

Aspect	Mean	Deviation	Min	Max	Median
#32	66.0%	0.0%	66.0%	66.0%	66.0%
#33	83.0%	24.04%	66.0%	100.0%	83.0%
#34	33.0%	46.67%	0.0%	66.0%	33.0%
#40	83.0%	24.04%	66.0%	100.0%	83.0%
#43	16.5%	23.33%	0.0%	33.0%	16.5%
#44	33.0%	46.67%	0.0%	66.0%	33.0%
Mean	**52.1%**	**13.39%**	**42.63%**	**67.88%**	**52.1%**
Deviation	**26.92%**	**16.69%**	**31.85%**	**18.67%**	**26.92%**
Min	**0.0%**	**0.0%**	**0.0%**	**33.0%**	**0.0%**
Max	**100.0%**	**46.67%**	**100.0%**	**100.0%**	**100.0%**
Median	**66.0%**	**0.0%**	**66.0%**	**66.0%**	**66.0%**

Summary

Number of instances:	43
Number of evaluated instances:	43
Number of instances above the threshold:	27
Precision:	62.79%
Total number of instances:	56
Total number of evaluated instances:	56
Total number of instances above the threshold:	32
Recall:	84.38%

5. ábra. A Bauhaus klón detektáló eszköz helyességének statisztikai mutatói

A BEFRIEND különböző nézeteket nyújt a felhasználói kiértékelések begyűjtéséhez, és az adatok elemzéséhez. Például a statisztikai adatok nézete a felhasználók által a különböző szempontok szerint felvitt értékelésekről nyújt átfogó információt (lásd 5. ábra).

A szerző hozzájárulása az eredményekhez

A szerző dolgozta ki a statisztikai módszerek alkalmazásának módját, amellyel a drill-down algoritmus hibák helyének beazonosítási képességét vizsgálta. Ő végezte el a statisztikai elemzést, kiértékelte és bemutatta az eredményeket. A költségbecslő modell empirikus validációját is a szerző végezte el, implementálta a támogató prototípus eszközöket, valamint elemezte és kiértékelte a kapott eredményeket. A szerző dolgozta ki a módszert, amellyel a tervezési minták alkalmazásának a szoftverek karbantarthatóságára gyakorolt hatását vizsgálta. Ő elemezte a JHotDraw rendszer egymást követő verzióit és a különböző tervezési minta benchmarkokban található rendszereket, valamint feldolgozta és kiértékelte a kísérleti eredményeket. Az általánosított testvér összekapcsoló algoritmuson kívül a szerző implementálta és mutatta be a BEFRIEND-et, ami egy általános benchmark visszatervező eszközök kiértékelésére.

Összefoglalás

A tézis főbb kutatási eredményei három tézispontba csoportosíthatók. Először megvizsgáltuk a jelenleg létező gyakorlati módszereket a szoftverek minőségének modellezésére, majd javasoltunk egy teljesen újszerű, valószínűség számításon alapuló megközelítést, ami sikeresen kiküszöböli a jelenlegi modellek legtöbb hátrányát szakértői vélemények integrálásával, valamint egy úgynevezett benchmark adatbázis használatával, amely a minőségi becslés alapjául szolgál. Az elért eredmények segítséget nyújthatnak például a menedzserek, vagy egyéb nem technikai szakemberek számára, hogy átfogó képet kapjanak a saját rendszerük karbantarthatóságáról. Egy Java nyelvre készített prototípus modell után kidolgoztunk egy C# karbantarthatósági modellt, amelyet sikeresen alkalmaztunk egy valós ipari környezetben. Az elkészített modell eredményei nagy mértékben tükrözték a szakértők véleményét. Az elért új kutatási eredmények gyakorlati alkalmazásának megkönnyítése érdekében a módszert implementáltuk, az elkészült eszköz kipróbálásra le is tölthető.

Annak érdekében, hogy megfelelően alacsony szintű, technikai információval lássuk el a fejlesztőket, akik javítani szeretnének a rendszer karbantarthatóságán a modellünk által szolgáltatott eredmények alapján, kidolgoztunk egy úgynevezett drill-down algoritmust, amely minden egyes forráskód elemhez egy karbantarthatósági indexet rendel a rendszer szintű karbantarthatóságon túl. Ezen mutató segítségével pontosan meghatározhatók azok a karbantarthatóság szempontjából legkritikusabb elemek, amelyek javítása jelentős rendszer szintű minőségi javulást okoznának. Továbbá ez az alacsony szintű információ segítségünkre lehet a hibára hajlamos részek kiszűrésében, a tesztelési erőforrások beosztásában vagy a kód átvizsgálások célpontjainak kijelölésében. Az elméleti eredmények mellett a kidolgozott modellek és módszerek szoftverevolúció során történő számos gyakorlati alkalmazását is bemutattuk. Empirikus esettanulmányok segítségével megmutattuk, hogy az általunk kidolgozott karbantarthatósági mutató jól jelzi a hibára hajlamos osztályokat egy rendszeren belül. Megalkottunk egy költségbecslő modellt is, amely egy rendszer karbantarthatóságának változása alapján nagy pontossággal előre tudja jelezni a jövőbeni fejlesztések költségeit.

Bemutattunk néhány olyan esettanulmányt is, ahol a karbantarthatóság és a forráskódban megtalálható tervezési minták kapcsolatát vizsgáltuk, mivel az általános nézet az, hogy a tervezési minták használata jobb minőségű kódot eredményez. Arra a következtetésre jutottunk, hogy a rendszerben lévő minták sűrűsége valóban erős korrelációt mutat a rendszer karbantarthatóságával. Ezen kísérleti eredmények csupán a kezdeti lépést jelentik az egyéb kódolási gyakorlatok és minták (mint például a tervezési minták, ellenminták vagy refaktorálás) vélt vagy valós karbantarthatóságra gyakorolt hatásának empirikus vizsgálata felé, amely elengedhetetlen ahhoz, hogy az üzleti beállítottságú résztvevőket is meggyőzzük a karbantarthatóságba fektetett energia megtérüléséről.

A 7. táblázatban összefoglaljuk az egyes tézispontokhoz kapcsolódó főbb publikációkat.

$\mathcal{N}o.$	[4]	[10]	[3]	[5]	[11]	[12]	[14]	[13]	[2]	[17]	[8]	[9]
I	•	•	•	•								
II			•	•	•	•	•					
III								•	•	•	•	•

7. táblázat. Az egyes tézispontokhoz kapcsolódó főbb publikációk

Köszönetnyilvánítás

Először is szeretnék köszönetet mondani témavezetőmnek, Dr. Ferenc Rudolfnak, hogy végigkísérte és segítette tanulmányaimat, illetve hogy sok, a kutatáshoz nélkülözhetetlen dolgot tanított nekem. Szintén szeretném megköszönni Dr. Gyimóthy Tibornak, a Szoftverfejlesztés Tanszék vezetőjének a kutatómunkám során nyújtott folyamatos támogatását. Külön köszönetem szeretném kifejezni Dr. Fülöp Lajos Jenőnek, akit második mentoromnak tekintek, és akitől nagyon sok motivációt és bátorítást kaptam a PhD tanulmányaim elején. Akiknek még sok köszönettel tartozom, mint kolléga és társszerző, azok Dr. Bakota Tibor, Dr. Beszédes Árpád, Dr. Siket István, Dr. Jász Judit, Dr. Schrettner Lajos, Dr. Günter Kniesel, Alexander Binun, Dr. Alexander Chatzigeorgiou, Dr. Yann-Gaël Guéhéneuc, Dr. Nikolaos Tsantalis, Kakuja-Tóth Gabriella, Ilia Árpád, Végh Ádám Zoltán, Körtvélyesi Péter, Ladányi Gergely, Bán Dénes, Kádár István, Faragó Csaba, Csaba Béla és Illés László. Szeretnék még külön köszönetet mondani David P. Curley-nek a disszertációm angol nyelvű változatának lektorálásáért. Meg kell említenem továbbá, hogy a kutatás a TÁMOP-4.2.4.A/2-11/1-2012-0001 Nemzeti Kiválóság Program című kiemelt projekt keretében zajlott. A projekt az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósul meg.

Hegedűs Péter, 2014. június

Hivatkozások

[1] *Marwen Abbes, Foutse Khomh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An Empirical Study of the Impact of Two Antipatterns, Blob and Spaghetti Code, on Program Comprehension. In* Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), *pages 181–190. IEEE, 2011.*

[2] *Tibor Bakota, Péter Hegedűs, Gergely Ladányi, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A Cost Model Based on Software Maintainability. In* Proceedings of the 28th IEEE International Conference on Software Maintenance (ICSM 2012), *pages 316–325, 2012.*

[3] *Tibor Bakota, Péter Hegedűs, István Siket, Gergely Ladányi, and Rudolf Ferenc. QualityGate SourceAudit: a Tool for Assessing the Technical Quality of Software. In* 2014 Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), *pages 440–445. IEEE, 2014.*

[4] *Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A Probabilistic Software Quality Model. In* Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM 2011), *pages 368–377, Williamsburg, VA, USA, 2011. IEEE Computer Society.*

[5] *Rudolf Ferenc, Péter Hegedűs, and Tibor Gyimóthy. Software Product Quality Models. In Tom Mens, Alexander Serebrenik, and Anthony Cleve, editors,* Evolving Software Systems, *pages 65–100. Springer Berlin Heidelberg, 2014.*

[6] *Francesca Arcelli Fontana, Andrea Caracciolo, and Marco Zanoni. DPB: A Benchmark for Design Pattern Detection Tools. In* Proceedings of the 16th European Conference on Software Maintenance and Reengineering (CSMR), *pages 235–244, 2012.*

[7] *Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts.* Refactoring: Improving the Design of Existing Code. *Addison-Wesley, 1999.*

[8] *Lajos Jenő Fülöp, Árpád Ilia, Ádám Zoltán Végh, Péter Hegedűs, and Rudolf Ferenc. Comparing and Evaluating Design Pattern Miner Tools.* ANNALES UNIVERSITATIS SCIENTIARUM DE ROLANDO EÖTVÖS NOMINATAE Sectio Computatorica, *XXXI:167–184, 2009.*

[9] *Lajos Jenő Fülöp, Péter Hegedűs, and Rudolf Ferenc. BEFRIEND – a Benchmark for Evaluating Reverse Engineering Tools.* Periodica Polytechnica Electrical Engineering, *52(3-4):153–162, 2008.*

[10] *Péter Hegedűs. A Probabilistic Quality Model for C# – an Industrial Case Study.* Acta Cybernetica, *21(1):135–147, 2013.*

[11] *Péter Hegedűs, Tibor Bakota, Gergely Ladányi, Csaba Faragó, and Rudolf Ferenc. A Drill-Down Approach for Measuring Maintainability at Source Code Element Level.* Electronic Communications of the EASST, *60:1–21, 2013.*

[12] *Péter Hegedűs, Tibor Bakota, László Illés, Gergely Ladányi, Rudolf Ferenc, and Tibor Gyimóthy. Source Code Metrics and Maintainability: a Case Study. In* Proceedings of the 2011 International Conference on Advanced Software Engineering & Its Applications (ASEA 2011), *pages 272–284. Springer-Verlag CCIS, 2011.*

[13] *Péter Hegedűs, Dénes Bán, Rudolf Ferenc, and Tibor Gyimóthy. Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability. In* Proceedings of the 2012 International Conference on Advanced Software Engineering & Its Applications (ASEA 2012), *pages 138–145. Springer-Verlag CCIS, 2012.*

[14] *Péter Hegedűs, Gergely Ladányi, István Siket, and Rudolf Ferenc. Towards Building Method Level Maintainability Models Based on Expert Evaluations. In* Computer Applications for Software Engineering, Disaster Recovery, and Business Continuity, *pages 146–154. Springer, 2012.*

[15] *ISO/IEC.* ISO/IEC 9126. Software Engineering – Product quality 6.5. *ISO/IEC, 2001.*

[16] *ISO/IEC.* ISO/IEC 25000:2005. Software Engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE. *ISO/IEC, 2005.*

[17] *Gergely Ladányi, Péter Hegedűs, Rudolf Ferenc, István Siket, and Tibor Gyimóthy. The Connection of the Bug Density and Maintainability of Classes. In* 8th International Workshop on Software Quality and Maintainability, *SQM, 2014 (presentation only). http://sqm2014.sig.eu/?page=program.*

[18] *William B. McNatt and James M. Bieman. Coupling of Design Patterns: Common Practices and Their Benefits. In* Proceedings of the 25th International Computer Software and Applications Conference on Invigorating Software Development, *COMPSAC '01, pages 574–579, Washington, DC, USA, 2001. IEEE Computer Society.*